TTIC 31010 / CMSC 37000 Algorithms, Winter Quarter 2019

Homework # 1 **Due: January 17, 2019**

Homeworks due in class on the due date. Please put your name on each page of your handin just in case pages get separated.

Lateness policy: up to 24 hours late: 10 points off. 24-48 hours late: 20 points off. More than 48 hours late: 60 points off (at this point, solutions will be posted - and you may look at them if you wish, but your answers should be in your own words).

Written homeworks are to be done *individually*. Group work is only for the oral-presentation assignments. If you have questions, please contact the course staff.

Please do *not* post questions or solutions on websites such as coursehero etc.

Problems:

(25 pts) 1. **Recurrences.** Solve the following recurrences, giving your answer in Θ notation. For each of them, assume the base case $T(x) = 1$ for $x \leq 10$. Show your work.

(a) $T(n) = T(n-3) + n^2$.

(b) $T(n) = T(n/2) + \lg n$.

(c) $T(n) = 5T(n-3)$.

(d) $T(n) = 2T(n/2) + (\lg n)^2$.

(25 pts) 2. **Find the problem.**

Recall the $\lg(n!)$ comparison-based sorting lower bound given in lecture. Suppose that instead of sorting the input, our goal is just to output a permutation of the input such that the smallest $n/4$ elements come before the next largest $n/4$ elements, which come before the next largest $n/4$ elements, which come before the top $n/4$ elements. Suppose we try to use that same lower-bound proof for sorting to give a $\lg(n!)$ lower bound for this problem. Where does the argument break down? Explain (e.g., include an example explaining where the argument breaks). Note that the argument had better break down since this problem can be solved in linear time.

(25 pts) 3. **Tight upper/lower bounds.** Consider the following problem.

INPUT: n^2 distinct numbers in some arbitrary order.

OUTPUT: an $n \times n$ matrix of the inputs having all rows and columns sorted in increasing order.

EXAMPLE: $n = 3$, so $n^2 = 9$. Say the 9 numbers are the digits $1, ..., 9$. Possible outputs include:

```
1 2 3        1 4 5        1 3 4
4 5 6   or   2 6 7   or   2 5 8  or  ...
7 8 9        3 8 9        6 7 9
```

It is clear that we can solve this problem in time $O(n^2 \log n)$ by just sorting the input (remember that $\log n^2 = O(\log n)$) and then outputting the first n elements as the first row, the next n elements as the second row, and so on. Your job in this problem is to prove a matching $\Omega(n^2 \log n)$ *lower bound* in the comparison-based model of computation.

For simplicity, you can assume n is a power of 2.

Some hints: Show that if you could solve this problem using $o(n^2 \log n)$ comparisons (in fact, in less than $n^2 \lg(n/e)$ comparisons), then you could use this to violate the $\lg(m!)$ lower bound for comparisons needed to sort m elements. You may want to use the fact that $m! > (m/e)^m$. Also, recall that you can merge two sorted arrays of size n using at most $2n - 1$ comparisons.

(25 pts) 4. **Divide-and-conquer.**

Consider the following problem.

INPUT: an array A of n numbers (not necessarily distinct) in some arbitrary order.

OUTPUT: an index i of a *local minimum* in A (see definition below).

(a) Define a local minimum to be an element $A[i]$ such that $A[i]$ is *strictly less* than both its neighbors (or its single neighbor if $i = 1$ or $i = n$); the algorithm must find some such entry or output that no local minimum exists. Note that it is possible for there not to exist a local minimum under this (non-standard) definition, e.g., if all entries in A are identical. Prove an $\Omega(n)$ lower bound for this problem in the query model. That is, give an adversary argument showing that any correct algorithm must query $\Omega(n)$ entries in A. You may assume for simplicitly that the algorithm is deterministic.

(b) Define a local minimum to be an element $A[i]$ such that $A[i]$ is *less than or equal to* each of its neighbors (or its single neighbor if $i = 1$ or $i = n$). It is not hard to see that such a local minimum must exist (think of placing a marble at some arbitrary initial location in the array and having it roll downhill). Give an algorithm for solving this problem that makes only $O(\log n)$ queries.